Data Visualization

Contents

1 Loading in packages

To start we will load in the tidyverse package that we installed last class.

To load a package (which is different than installing a package), we use the `library()`

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

You only need to install a package once, but you need to reload it every time you start a new session.

If you got the error message "there is no package called 'tidyverse'", you'll need to first install it (using `install.packages("tidyverse")`), then run `library(tidyverse)` once again.

2 Accessing data

Mostly, data that you will use will come from external data that we will specifically load into `R`. We will discuss the process importing and exporting data in class 5.

Until then, we'll use data that is already built into `R` or one of the `R` packages.

Since `tidyverse` is now loaded into `R`, we can now access the `mpg` dataset

This dataset is what is called a **data frame**. Data frame's are one of the most popular types of data objects in `R`, but they are not the only version. In this class, we will mostly use data frame's as they are used by most of `R`'s modeling software.

```
mpg
```

```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 224 more rows
```

Among the variables in `mpg` are:

1. `displ`, a car's engine size, in liters.
2. `hwy`, a car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

To learn more about `mpg`, open its help page by running `?mpg`.

To see the entire dataset run

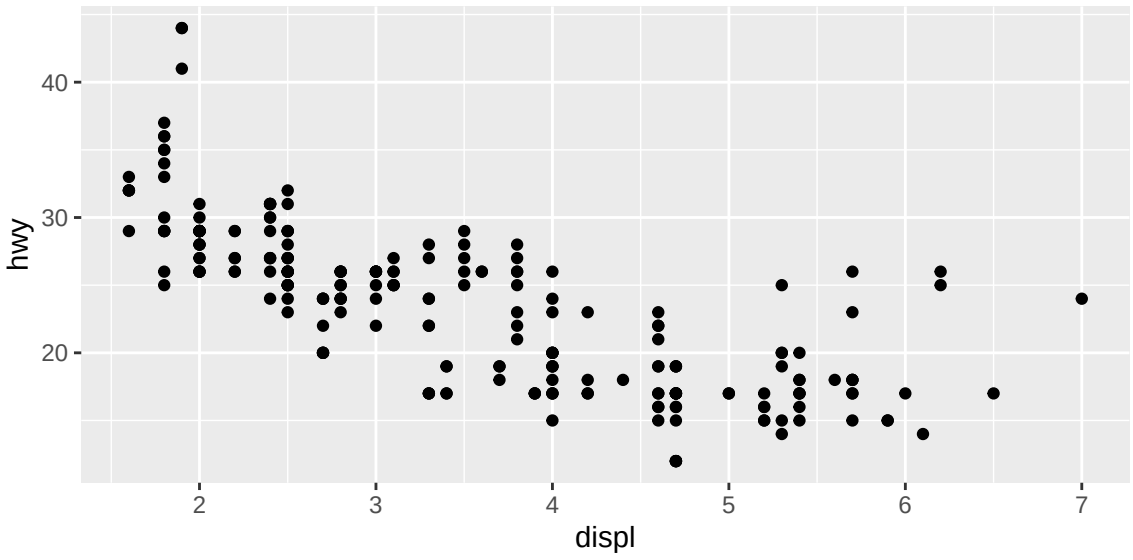
```
View(mpg)
```

3 Creating a ggplot

Below, we'll explore the question "Do cars with big engines use more fuel than cars with small engines?"

To explore this question, we'll first look at a scatter plot of `displ` on the x-axis and `hwy` on the y-axis:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



It appears that cars with smaller engine size tend to get more miles per gallon on the highway.

What did we do:

- With ggplot2, you begin a plot with the function `ggplot()`.
- `ggplot()` creates a coordinate system that you can add layers to. The first argument of `ggplot()` is the dataset to use in the graph. So `ggplot(data = mpg)` creates an empty graph (not very interesting).

```
ggplot(data = mpg)
```

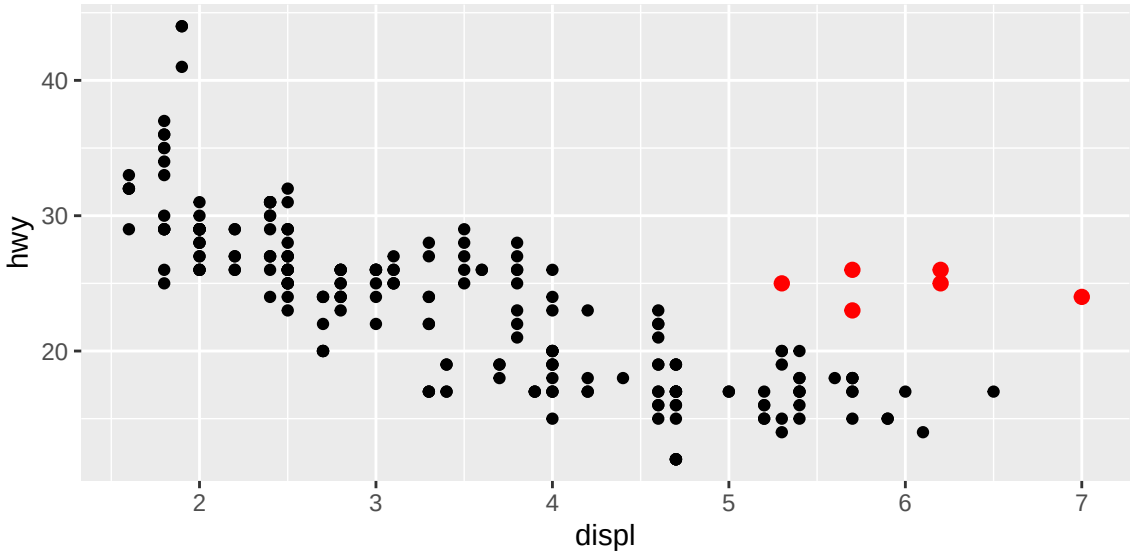

- Then, we added a layer to our `ggplot()`.
- The function `geom_point()` adds a layer of points to your plot, which creates a scatterplot.
- The `aes()` function is one you will frequently use with `ggplot()`.
- `aes()` is a `mapping` function that defines how variables in your dataset are mapped to visual properties of your plot.
  - Here, the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes.

In general, a `ggplot` will look something like this:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

3.1 Fancifying a ggplot

Notice that some of the cars fall outside of the normal trend

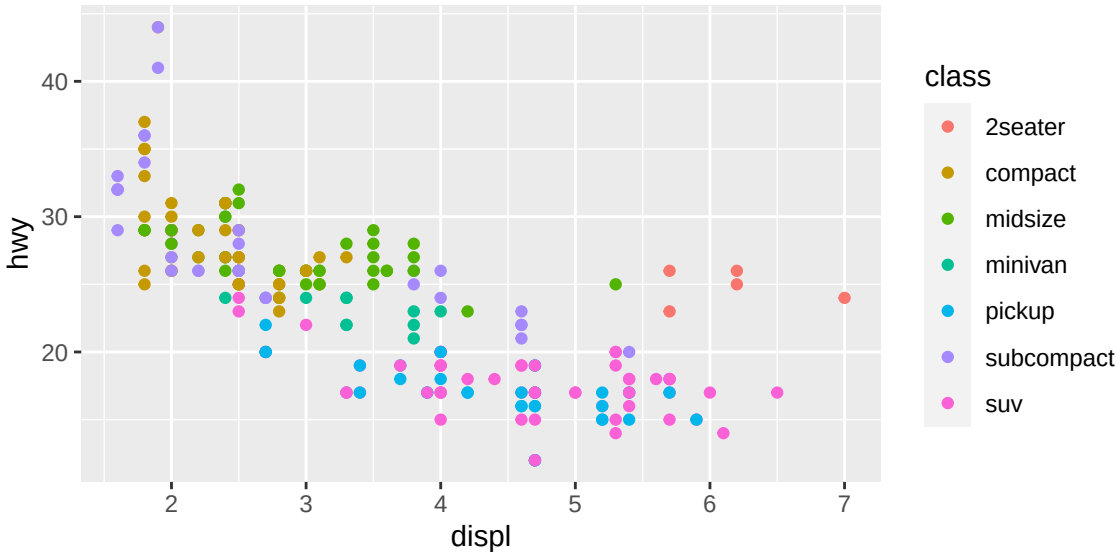


Is it possible that these cars have a different `class` than the other cars?

Let's check this out. We want to look at the trend of `displ` and `hwy` as we did before, but we also want to include what `class` the cars are.

One way to do this is to is to make the points different colors based on what `class` they are. This can be done using the same code as before, but adding the `color` arguement to the `aes()` function.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

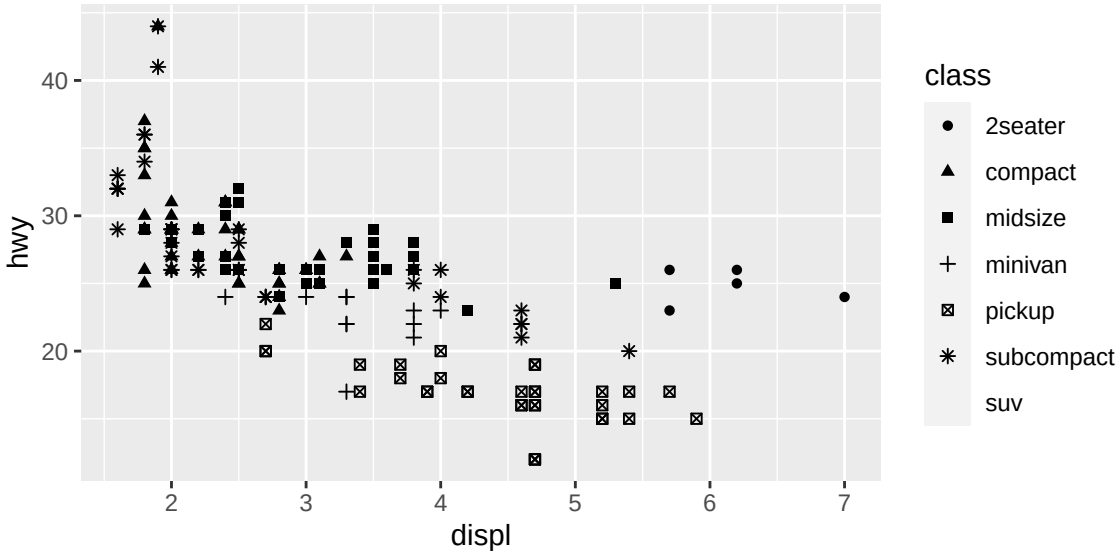


Another way is to change the `shape` of the points for cars of different classes:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

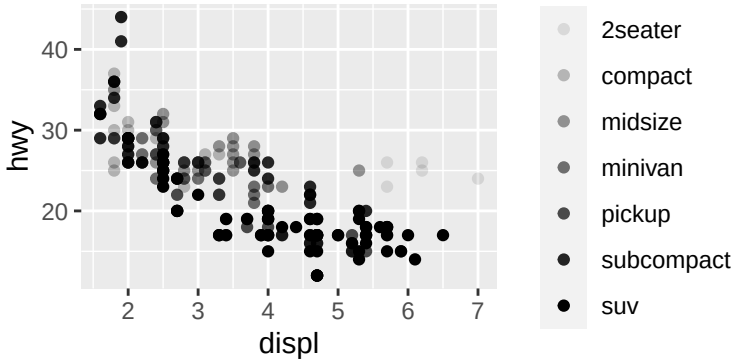
```
## Warning: Removed 62 rows containing missing values (geom_point).
```



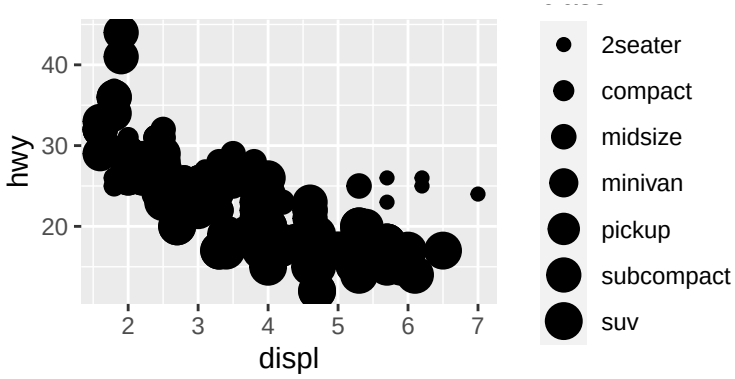
What happened to the SUVs? `ggplot2` will only use six shapes at a time. By default, additional groups will go unplotted when you use the shape aesthetic (though there are ways to override this).

Other options are to change `alpha` aesthetic, which controls the transparency of the points, or the `size` aesthetic, which controls the size of the points.

```
# Left
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```



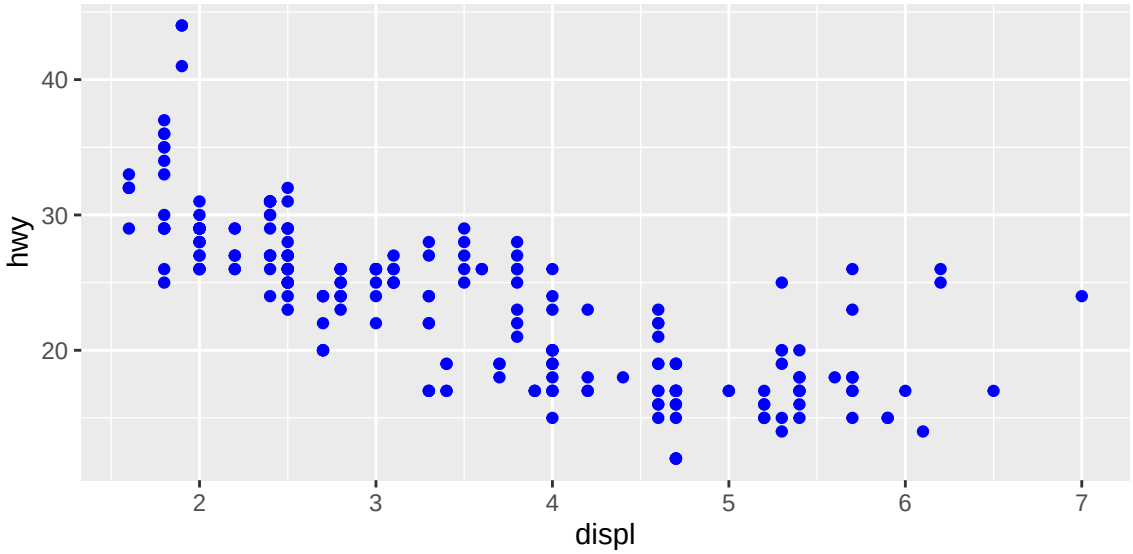

```
# Right
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
```



For each aesthetic, you use `aes()` to associate the name of the aesthetic with a variable to display.

You can also *set* the aesthetic properties of your geom manually. For example, we can make all of the points in our plot blue:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



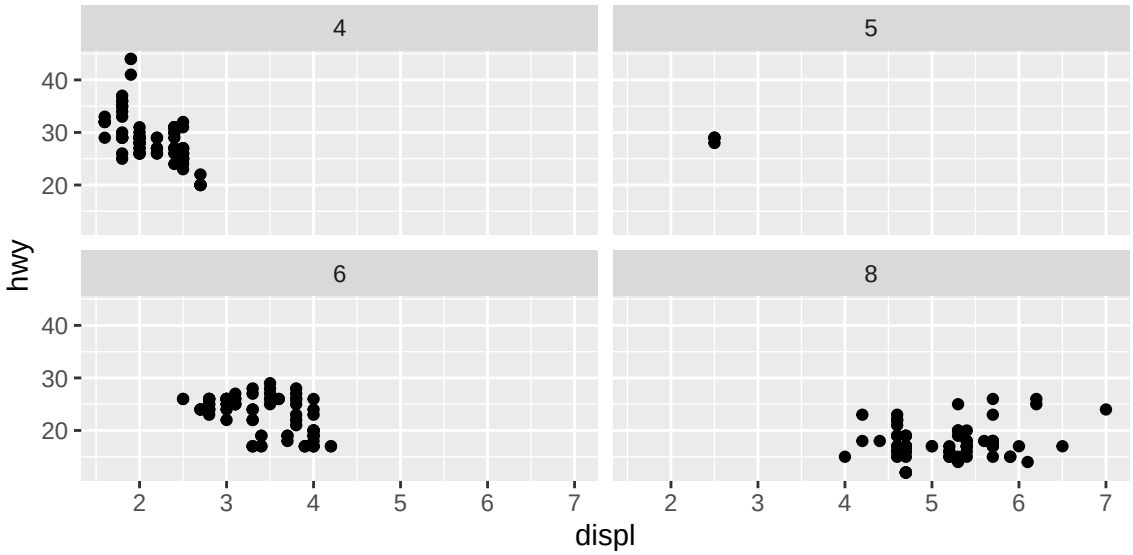
Notice the blue is outside of the `aes()` function.

4 Facet plotting

Above, we added the third variable `class` using different aesthetics. Another option is to create different plots for each level of another variable. To do this we can use `facet_wrap()`.

For example, let's take the first plot we started with and make unique plots for each number of cylinders (`cyl`) the cars have.

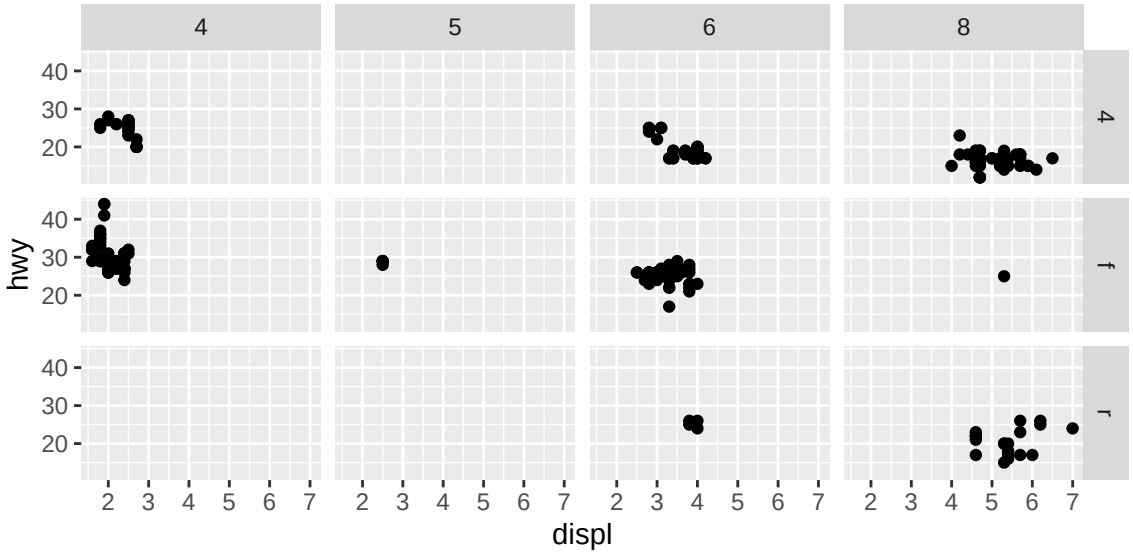
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~cyl)
```



If we wanted to complete the above use two variables (instead of one), we can use `facet_grid()`.

Here, we'll repeat the `cyl` plot but show it for each type of drive train (`drv` where f = front-wheel drive, r = rear wheel drive, 4 = 4wd).

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```



5 Going further

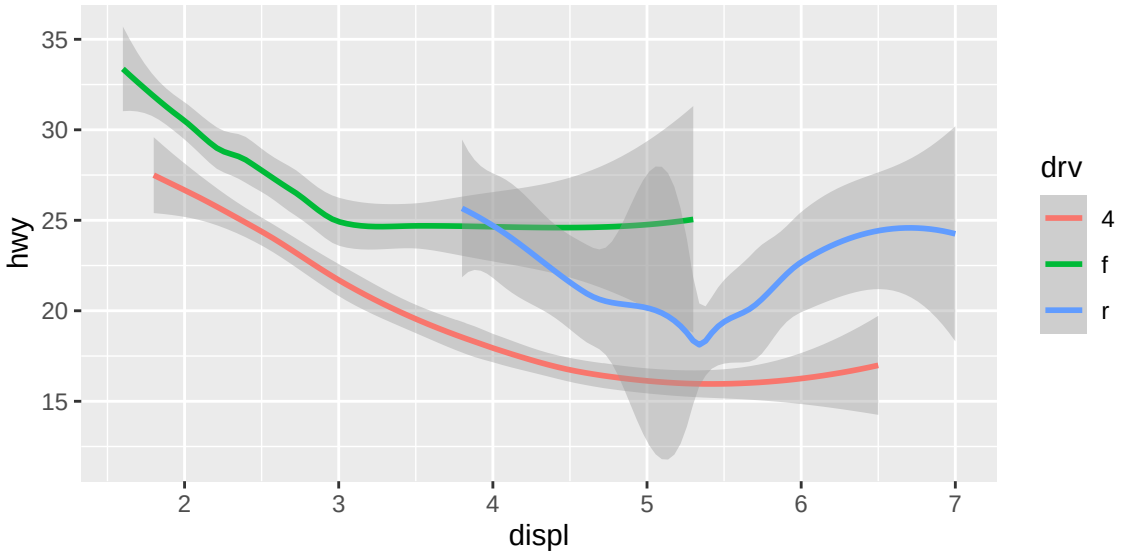
The `ggplot2` package contains over 40 geoms, each which will provide different ways of looking at your data (see https://exts.ggplot2.tidyverse.org/gallery/ for a sampling).

The best way to get a comprehensive overview is the ggplot2 cheatsheet, which you can find at http://rstudio.com/resources/cheatsheets.

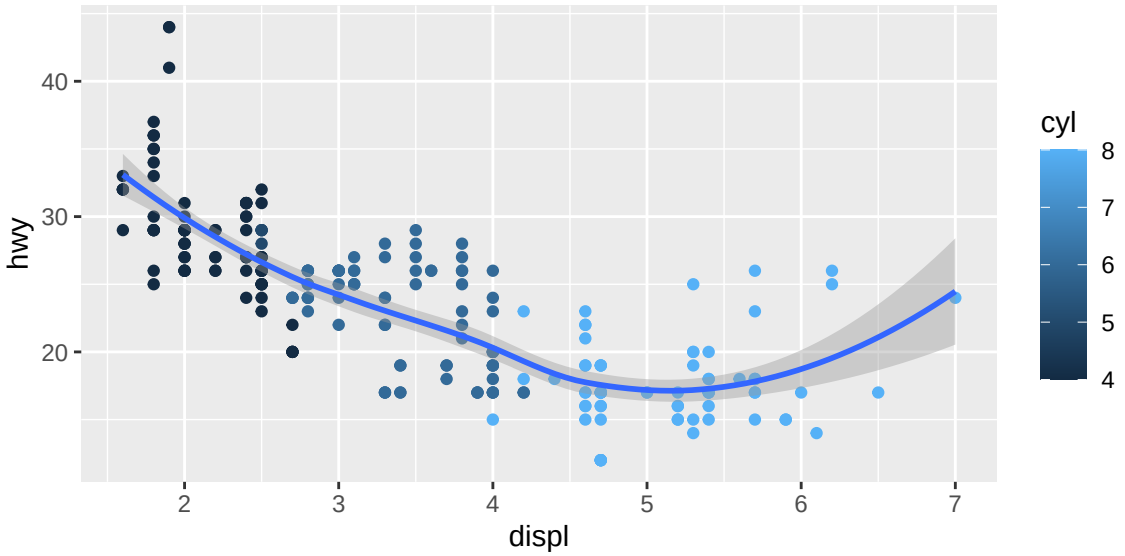
To learn more about any single geom, use the help (e.g. `?geom_point`).

Here's a sample of some of the options:

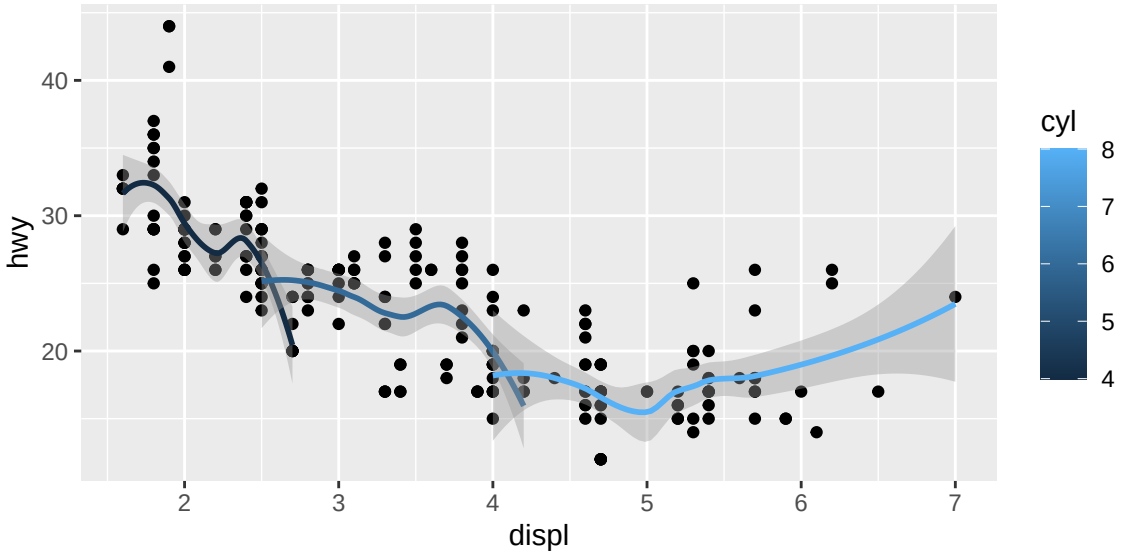
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, color = drv))
```




```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = cyl)) +
  geom_smooth()
```




```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(mapping = aes(group = cyl, color = cyl))
```




```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy,  group = cyl, color = cyl)) +
  geom_point() +
  geom_smooth()
```